

# ***New Methods in Parallelism***

## ***Getting Performance on the Origin 2000***

***Briefing to ARL/NRL***

***January 27, 1998***

***Jim Taft  
jtaft@nas.nasa.gov***

# ***Motivation for Parallel Computing***

- Continuing demand to solve ever more compute intensive problems**
- Need parallel because we are approaching physical limits of single CPU**
- Extremely cost effective if scaling can be achieved with commodity parts**
- Very attractive to vendors as large systems can be built from small ones**
- We really have no choice. It is the only hardware option of the future.**

# ***Why Origin 2000?***

- 1) ARC3D test results (6.4 GFLOPS)**
- 2) GEOS GCM test results (Faster than 8 CPU J90)**
- 3) Matrix Factorization test results (3.1 GFLOPS)**
- 4) Compute Server Suite test results (45 MFLOPS – best)**
- 5) NPB 2.0 test results ( excellent scaling – some 32x on 32 CPUs)**
- 6) Throughput test results (100s of benchmarks)**
- 7) Architecturally advanced**
  - 1) 1 of 2 shared memory NUMA based SMP systems**
  - 2) Allows both shared memory and message passing work**
  - 3) Full single system 64 bit UNIX OS (large address space)**
  - 4) High performance single image I/O**
  - 5) Scalable to much larger CPU and memory counts**

# ***NAS Origin 2000 Project – Goals***

---

- Prove Legacy Code can Port Successfully**
  - 1) Low level of effort**
  - 2) High performance possible**
  
- Develop Standard Optimization techniques**
  - 1) Maintain Cray vector performance**
  - 2) Achieve high performance on the new systems**
  - 3) Achieve result in a single code (test bed = OVERFLOW)**
  
- Develop New Multi-Level Parallelism Concept**
  - 1) Implement in a production code (test bed = OVERFLOW)**
  - 2) Develop a standard library for the user community**
  
- Develop Cooperative Agreements with other sites**
  - 1) DAO**
  - 2) LaRC**
  - 3) Commercial Customers**

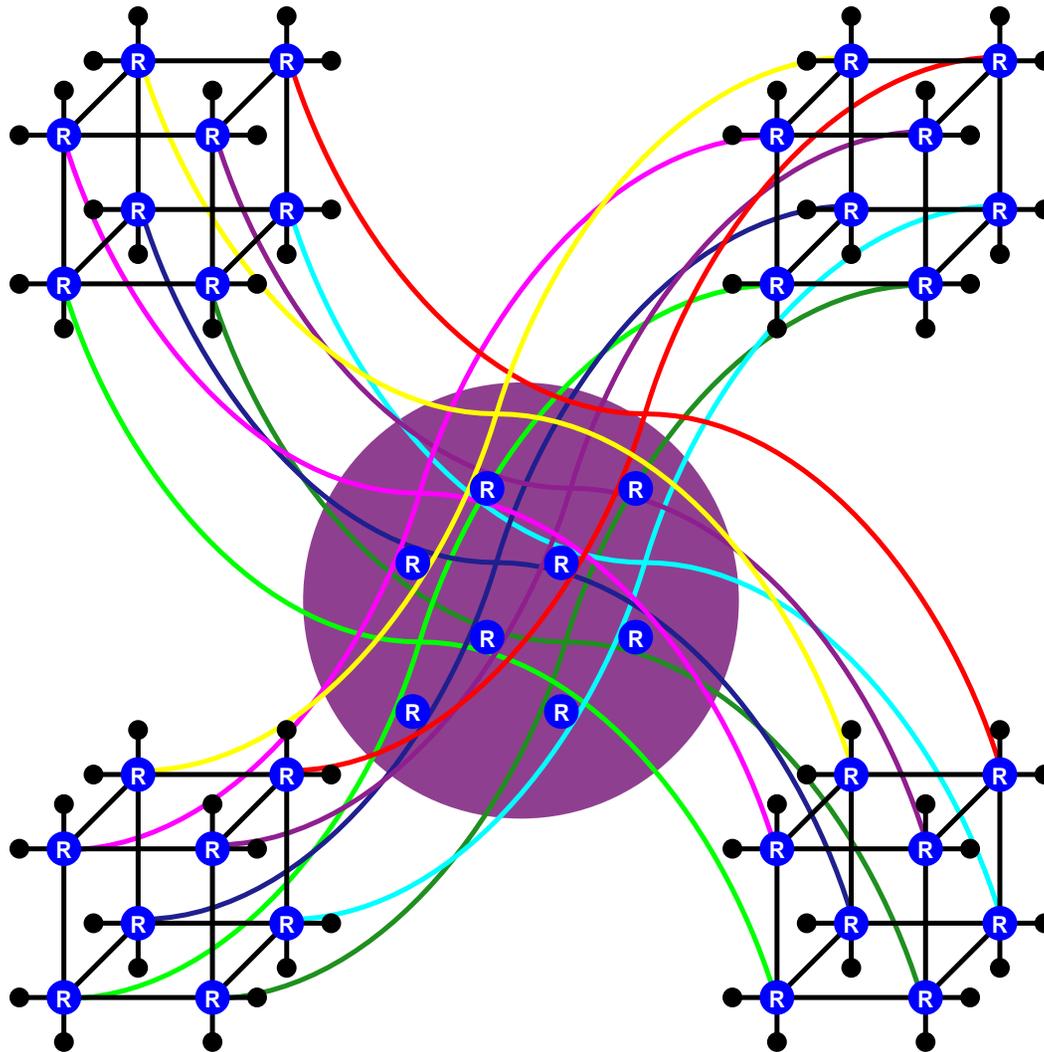
# ***NASA AMES/NAS Origin 2000 Project – Background***

- Effort started with concerns about RISC systems**
  - Many failures in highly touted past systems**
- First effort focussed on ARC3D performance**
  - ARC3D chosen because it ran 4% of C90 on one R8000 CPU**
  - ARC3D was indicative of solvers in use at NAS**
- Second Effort was focussed on OVERFLOW 1.7**
  - OVERFLOW chosen because it is true production code**
  - OVERFLOW was indicative of "toughest" case at NAS**

# *Origin 2000 (Turing) System*

- **Cost < \$2.5M**
- **Configuration:**
  - 64 CPUs
  - 16 GB of main memory
  - 330 GB of disk
    - system has 2 scratch partitions of 50 GB
    - each scratch partition runs about 50 MB/sec
- **Status:**
  - Systems are up and stable (30 day uptimes under load)
  - Load averages 50–60 CPUs out of 64 – excellent

# Origin 2000 – The Architecture



- 128 CPU system
- Connection via bristled hypercube
- Five hops worst case access

# *Part I*

# *ARC3D*

# ***ARC3D – Classic Parallelism***

**There are two classic ways of doing parallel work on existing systems**

- Fine grained**
  - This is usually termed compiler parallelism**
- Coarse grained**
  - Coarse grained by compiler directives**
  - Coarse grained by code conversion to Message Passing (MPI)**

# ***ARC3D – Project Background***

## **Phase I – Single CPU Optimizations**

- Target was 30% of single C90 CPU performance**
- Engineering approach to be used – not Computer Science**
  - Keep code portable**
  - Only allow "reasonable" code changes**
  - Emphasis on rapid port**
- Target was O2000 CPU to be 30% of single C90 CPU**
- Achieved this single CPU result in two weeks**

## **Phase II– Multiple CPU Optimizations**

- Target was full C90 Performance**
- Achieved this result on 64 CPUs in two months**

**All ARC3D work used coarse and fine grained compiler parallelism**

# ***ARC3D – Single CPU Results***

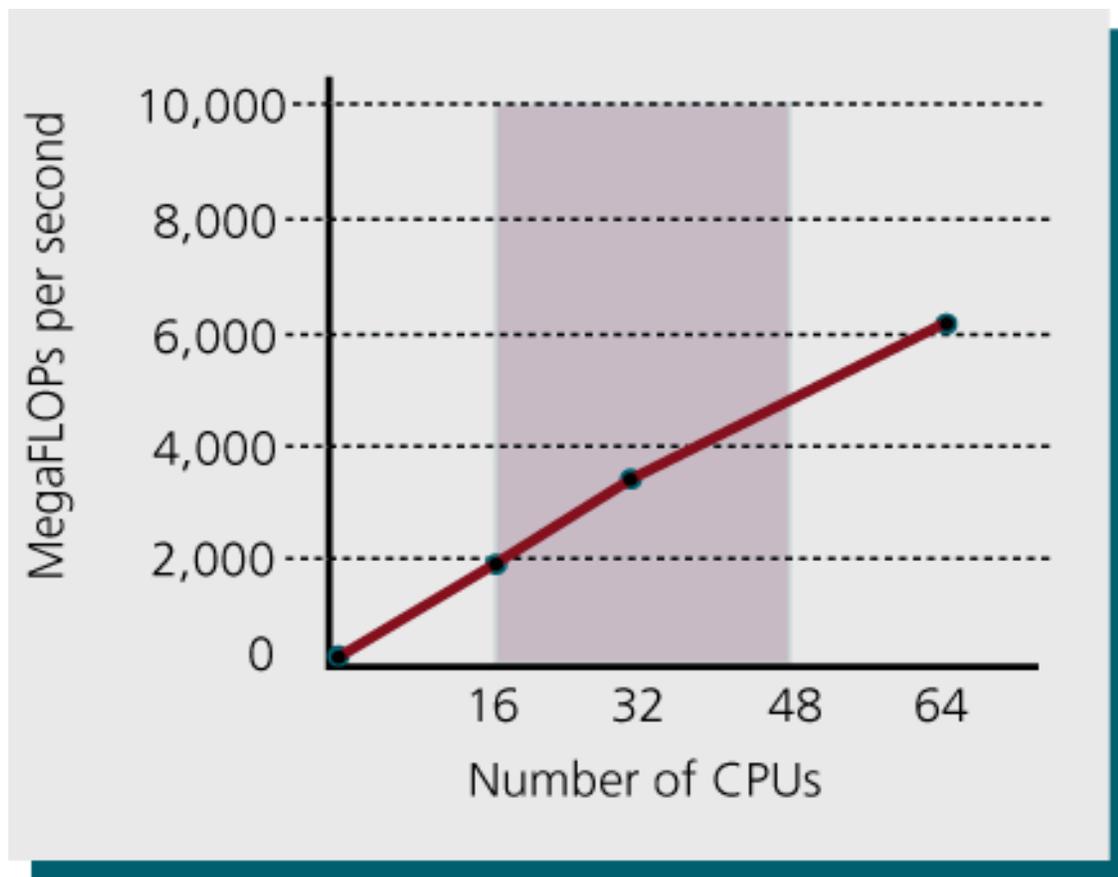
**Problem: 192 x 192 x 192 (7.1M points)**

<b><u>System Description:</u></b>	<b><u>1 CPU (sec)</u></b>	<b><u>4 CPU (sec)</u></b>	<b><u>8 CPU (sec)</u></b>
<b>C90</b>	<b>23</b>		
<b>T90</b>	<b>15</b>		
<b>O2000</b>	<b>72</b>	<b>23</b>	<b>13</b>

## **Notes:**

- These are old parallel results – no data distribution awareness**
- Even so, 8 CPUs outperform single T90 CPU**
- This phase completed in two weeks**
- Times are in seconds of elapsed wall clock time**

# ARC3D – 64 CPU Parallel Scaling Results



# ARC3D – 64 CPU Parallel Results

Routine	1-CPU Wall	64-CPU Wall	Speedup	Efficiency	% of Work	MFLOPs/sec
TOTAL	364.47	7.45	48.93	76.5%	100.0%	6309.8
step	13.83	0.60	23.05	36.0%	8.1%	----
xdirect	83.66	1.66	50.39	78.7%	22.3%	6914.5
ydirect	88.14	1.51	58.20	90.9%	20.3%	7580.4
zdirect	110.06	1.97	55.87	87.3%	26.4%	5828.3
rhsx	15.92	0.40	40.20	62.8%	5.3%	4813.7
rhsy	18.86	0.44	42.62	66.6%	5.9%	4308.1
rhsz	33.27	0.69	47.88	74.8%	9.3%	2743.4
bc	0.75	0.17	4.40	6.9%	2.3%	----

- 1) This parallel effort required an additional two months
- 2) Performance with this code could be further enhanced

# ARC3D – Step Routine

```
subroutine step
c*****
c*
c*****
#include "common.blk"

dimension sum(md)

call bc

c-----calculate rhs and smoothing
c$doacross local(k)
do 120 k=2,km
call rhsz(k)
120 continue
c$doacross local(l)
do 130 l=2,lm
call rhsy(l)
130 continue
c$doacross local(l)
do 140 l=2,lm
call rhsx(l)
140 continue

c-----calculate the residual
resid=0.0
do 109 l=2,lm
sum(l)=0.0
109 continue
c$doacross local(j,k,l)
do 110 l=2,lm
do 110 k=2,km
do 110 j=2,jm
sum(l)=sum(l)+s(1,j,k,l)**2+s(2,j,k,l)**2+s(3,j,k,l)**2
.
+s(4,j,k,l)**2+s(5,j,k,l)**2
110 continue
do 111 l=2,lm
resid=resid+sum(l)
111 continue
resid=resid/((jm-1)*(km-1)*(lm-1))
resid=sqrt(resid)/(dt+0.00005)
write(6,112) nc,resid
112 format('0','n= ',i5,3x,'l2 residual =' ,1p,e16.8)
180 waltim(03)=waltim(03)+second(xx)

c-----scale s
waltim(03)=waltim(03)-second(xx)
c$doacross local(j,k,l)
do 190 l=2,lm
do 190 k=2,km
do 190 j=2,jm
s(1,j,k,l)=s(1,j,k,l)*vardt(j,k,l)
s(2,j,k,l)=s(2,j,k,l)*vardt(j,k,l)
s(3,j,k,l)=s(3,j,k,l)*vardt(j,k,l)
s(4,j,k,l)=s(4,j,k,l)*vardt(j,k,l)
s(5,j,k,l)=s(5,j,k,l)*vardt(j,k,l)
190 continue
waltim(03)=waltim(03)+second(xx)

c-----x direction
waltim(04)=waltim(04)-second(xx)
c$doacross local(l)
do 200 l=2,lm
call xdirect(2,jm,2,km,l)
200 continue
waltim(04)=waltim(04)+second(xx)

c-----y direction
waltim(05)=waltim(05)-second(xx)
c$doacross local(l)
do 300 l=2,lm
call ydirect(2,km,2,jm,l)
300 continue
waltim(05)=waltim(05)+second(xx)

c-----z direction
waltim(06)=waltim(06)-second(xx)
c$doacross local(k)
do 400 k=2,km
call zdirect(2,lm,2,jm,k)
400 continue
waltim(06)=waltim(06)+second(xx)

c-----update
waltim(03)=waltim(03)-second(xx)
c$doacross local(j,k,l)
do 50 l=2,lm
do 50 k=2,km
do 50 j=2,jm
q(1,j,k,l)=q(1,j,k,l)+s(1,j,k,l)
q(2,j,k,l)=q(2,j,k,l)+s(2,j,k,l)
q(3,j,k,l)=q(3,j,k,l)+s(3,j,k,l)
q(4,j,k,l)=q(4,j,k,l)+s(4,j,k,l)
q(5,j,k,l)=q(5,j,k,l)+s(5,j,k,l)
50 continue
waltim(03)=waltim(03)+second(xx)

return
end
```

# ARC3D – Zdirect Routine

```

subroutine zdirect(ls,le,js,je,k)
c*****
c*
c*****
#include "common.blk"

real a(5,5,md),b(5,5,md),c(5,5,md),d(5,5),bb(5,5)

real l11,l21,l31,l41,l51,l22,l32,l42,l52,l33,l43,l53,l44,l54,l55

real t(5,md),e(6,md),f(md)

gam2=2.0-gamma
smuim1=smuim*dt

b1=2.0*smuim1
r4=0.0

c-----outer loop over L
do 900 j=js,je

c*****
c*   preload
c*****
c-----preload 3D values
do 100 l=1,lmax
t(1,l)=s(1,j,k,l)
t(2,l)=s(2,j,k,l)
t(3,l)=s(3,j,k,l)
t(4,l)=s(4,j,k,l)
t(5,l)=s(5,j,k,l)
f(1)=vardt(j,k,l)
e(1,l)=q(1,j,k,l)
e(2,l)=q(2,j,k,l)
e(3,l)=q(3,j,k,l)
e(4,l)=q(4,j,k,l)
e(5,l)=q(5,j,k,l)
e(6,l)=q(6,j,k,l)
100 continue

c-----preload coefficients
do m=1,5
do n=1,5
b(m,n,1)=0.0
a(m,n,1)=0.0
a(m,n,2)=0.0
bb(m,n)=0.0
enddo
enddo

```

```

c*****
c*   Coefficients
c*****
do 200 l=ls,le

r1=zzx(1,j,k)*hdz
r2=zzy(1,j,k)*hdz
r3=zzz(1,j,k)*hdz
r=e(1,l)
u=e(2,l)/r
v=e(3,l)/r
w=e(4,l)/r
sq1=smuim1*(e(6,l)/e(6,l+1))
sq2=smuim1*(e(6,l)/e(6,l-1))
uu=r1*u+r2*v+r3*w
ut=u*u+v*v+w*w
c1=0.5*gami*ut
c2=gamma*e(5,l)/r

d(1,1)=r4
d(1,2)=r1
d(1,3)=r2
d(1,4)=r3
d(1,5)=0.0
d(2,1)=r1*c1-u*uu
d(2,2)=r4+uu+r1*gam2*u

.....

d(5,3)=r2*(c2-c1)-gami*v*uu
d(5,4)=r3*(c2-c1)-gami*w*uu
d(5,5)=r4+gamma*uu

.....

do n=1,5
do m=1,5
c(m,n,l-1)=d(m,n)
enddo
enddo

a(1,1,l+1)=a(1,1,l+1)-sq1
a(2,2,l+1)=a(2,2,l+1)-sq1
a(3,3,l+1)=a(3,3,l+1)-sq1
a(4,4,l+1)=a(4,4,l+1)-sq1
a(5,5,l+1)=a(5,5,l+1)-sq1

c(1,1,l-1)=c(1,1,l-1)-sq2
c(2,2,l-1)=c(2,2,l-1)-sq2
c(3,3,l-1)=c(3,3,l-1)-sq2
c(4,4,l-1)=c(4,4,l-1)-sq2
c(5,5,l-1)=c(5,5,l-1)-sq2
200 continue

```

# ARC3D – Zdirect Routine

```
C*****
C*   Factor   *
C*****
  do 600 l=ls,le
    v1=f(1)
    bb(1,1)=b1*v1+1.0
    bb(2,2)=b1*v1+1.0
    bb(3,3)=b1*v1+1.0
    bb(4,4)=b1*v1+1.0
    bb(5,5)=b1*v1+1.0
    do m=1,5
      do n=1,5
        b(m,n,1)=bb(m,n)-v1*a(m,1,1)*b(1,n,1-1)
        .           -v1*a(m,2,1)*b(2,n,1-1)
        .           -v1*a(m,3,1)*b(3,n,1-1)
        .           -v1*a(m,4,1)*b(4,n,1-1)
        .           -v1*a(m,5,1)*b(5,n,1-1)
        t(m,1)=t(m,1)-v1*a(m,n,1)*t(n,1-1)
      enddo
    enddo

    l11=1.e0/b(1,1,1)
    u12=b(1,2,1)*l11
    l21=b(2,1,1)
    l22=1.e0/(b(2,2,1)-l21*u12)
    u23=(b(2,3,1)-l21*u13)*l22
    l53=b(5,3,1)-l51*u13-l52*u23
    l54=b(5,4,1)-l51*u14-l52*u24-l53*u34
    l55=1.e0/(b(5,5,1)-l51*u15-l52*u25-l53*u35-l54*u45)
    . . . . .
    d1=t(1,1)*l11
    d2=(t(2,1)-l21*d1)*l22
    d3=(t(3,1)-l31*d1-l32*d2)*l33
    d4=(t(4,1)-l41*d1-l42*d2-l43*d3)*l44
    d5=(t(5,1)-l51*d1-l52*d2-l53*d3-l54*d4)*l55

    t(5,1)=d5
    t(4,1)=d4-u45*d5
    t(3,1)=d3-u34*t(4,1)-u35*d5
    t(2,1)=d2-u23*t(3,1)-u24*t(4,1)-u25*d5
    t(1,1)=d1-u12*t(2,1)-u13*t(3,1)-u14*t(4,1)-u15*d5

    do 40 n=1,5
      c1=(c(1,n,1)*v1)*l11
      c2=(c(2,n,1)*v1-l21*c1)*l22
      c3=(c(3,n,1)*v1-l31*c1-l32*c2)*l33
      c4=(c(4,n,1)*v1-l41*c1-l42*c2-l43*c3)*l44
      c5=(c(5,n,1)*v1-l51*c1-l52*c2-l53*c3-l54*c4)*l55

      b(5,n,1)=c5
      b(4,n,1)=c4-u45*c5
      b(3,n,1)=c3-u34*b(4,n,1)-u35*c5
      b(2,n,1)=c2-u23*b(3,n,1)-u24*b(4,n,1)-u25*c5
      b(1,n,1)=c1-u12*b(2,n,1)-u13*b(3,n,1)-u14*b(4,n,1)-u15*c5
    40 continue
  600 continue
```

```
C*****
C*   Backsolve *
C*****
  do 800 l=le-1,ls,-1
    do 800 m=1,5
      t(m,1)=t(m,1)-b(m,1,1)*t(1,1+1)
      .           -b(m,2,1)*t(2,1+1)
      .           -b(m,3,1)*t(3,1+1)
      .           -b(m,4,1)*t(4,1+1)
      .           -b(m,5,1)*t(5,1+1)
    800 continue
  do 820 l=ls,le-1
    s(1,j,k,1)=t(1,1)
    s(2,j,k,1)=t(2,1)
    s(3,j,k,1)=t(3,1)
    s(4,j,k,1)=t(4,1)
    s(5,j,k,1)=t(5,1)
  820 continue
  900 continue

  return
end
```

# *Part II*

# *OVERFLOW*

# ***Why choose OVERFLOW***

**"I am quite concerned that benchmarks for stripped down elements of a CFD code are being confused with performance numbers for a full CFD code... which are usually 2–10 times slower."**

**"Taking these two things into account, my bet is that we will not beat a C-90 (e.g., 4–6 gflops) on a real application for a very long time (e.g., at least 3 years) on any SNX architecture. I would love to be proved wrong, but.... there has been the CM2s, the Hypercubes, the Paragons, the DaVincis, the SP1s, the SP2s, etc..... all of which were suppose to outperform the Crays on real problems, and which never really got close...."**

**"Please convince me otherwise...."**

**Crusty old CFDer**

# ***MLP – The New Parallelism***

**New system architectures are allowing the exploration of new and simpler methods of parallel processing. No longer is it necessary to make the move to MPI, with all of the uncertainty that this entails.**

**The New Parallelism is often called Multi-Level Parallelism (MLP) by vendors of shared memory architectures. It is differentiated from Message Passing in that all communication is through true shared memory references between independent processes. As a result latencies are on the order of hundreds of nanoseconds instead of tens of microseconds.**

**While it is being talked about, no one has implemented such a scheme in the real world. There is considerable doubt on how to do it properly. In many circles it is still a "pipe dream" in some distant future.**

**The results that follow are the first implementation of MLP for a production CFD code anywhere. The results show the exciting potential for this new methodology.**

# ***Opportunities for MLP – Evolution of CFD Codes***

**Most CFD codes are moving to Multi-zone for large problems:**

- A simpler method for solving larger problems**
- A simpler method of organizing the computation**
- A simpler method of building large grids**
- A method offering more hope of coarse grained parallelism**
- A method that is less sensitive to the system architecture**

**By their very nature, Multi-zone CFD codes are ideal candidates for MLP**

# ***The New Parallelism – Method***

**Overall philosophy behind the method is to decompose the problem as if you were going to build an MPI code – just don't use MPI**

- 1 – Group zones so they can be solved by parallel processes**
- 2 – Use global shared memory to communicate – not messages**
- 3 – Synchronize using barriers as needed**
- 4 – Keep life simple and only apply method to the core work**

**This last item is very important as it dramatically reduces the burden of the conversion effort when compared to MPI.**

# ***OVERFLOW and Multi-Level Parallelism (MLP)***

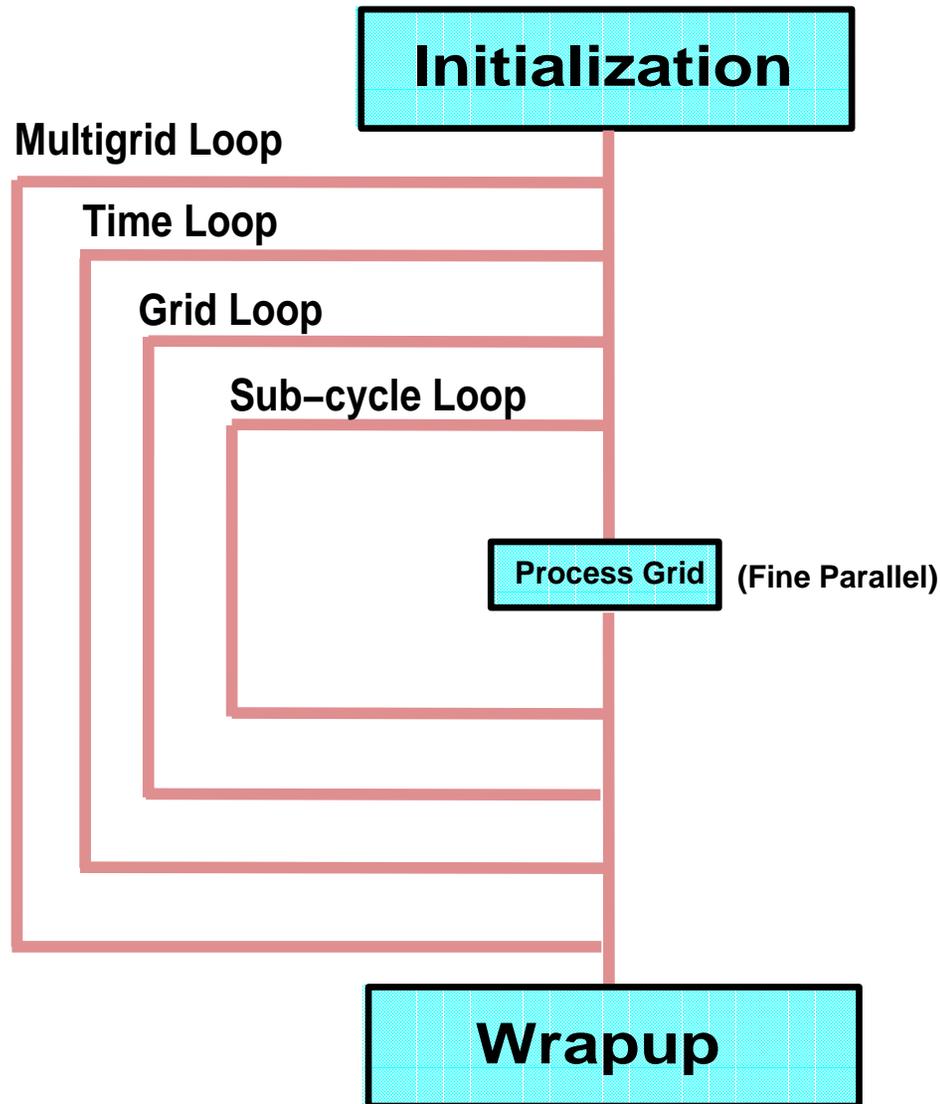
## **The Recipe:**

- Start with standard Shared Memory (C90) code**
- Assign needed global data to Shared Memory Arena**
- Setup and initialize as before on C90**
- Assign grids to groups**
- Spawn processes to process grid groupings**
- Perform work over groups of grids in parallel**
- Use compiler parallelism within a group**
- Perform dynamic load leveling after each time step**
- Wrapup as before on C90**

## **The Result:**

- Keeps code portable**
- Maintains Cray C90 performance level on Cray**
- Only requires "reasonable" code changes (~200 lines)**
- Very rapid port (weeks not years)**

# OVERFLOW – Logic Flow (C90 code)



## Characteristics:

- Grid initialization serial
- Grid processing serial
- Single Grid parallel
- Wrapup serial

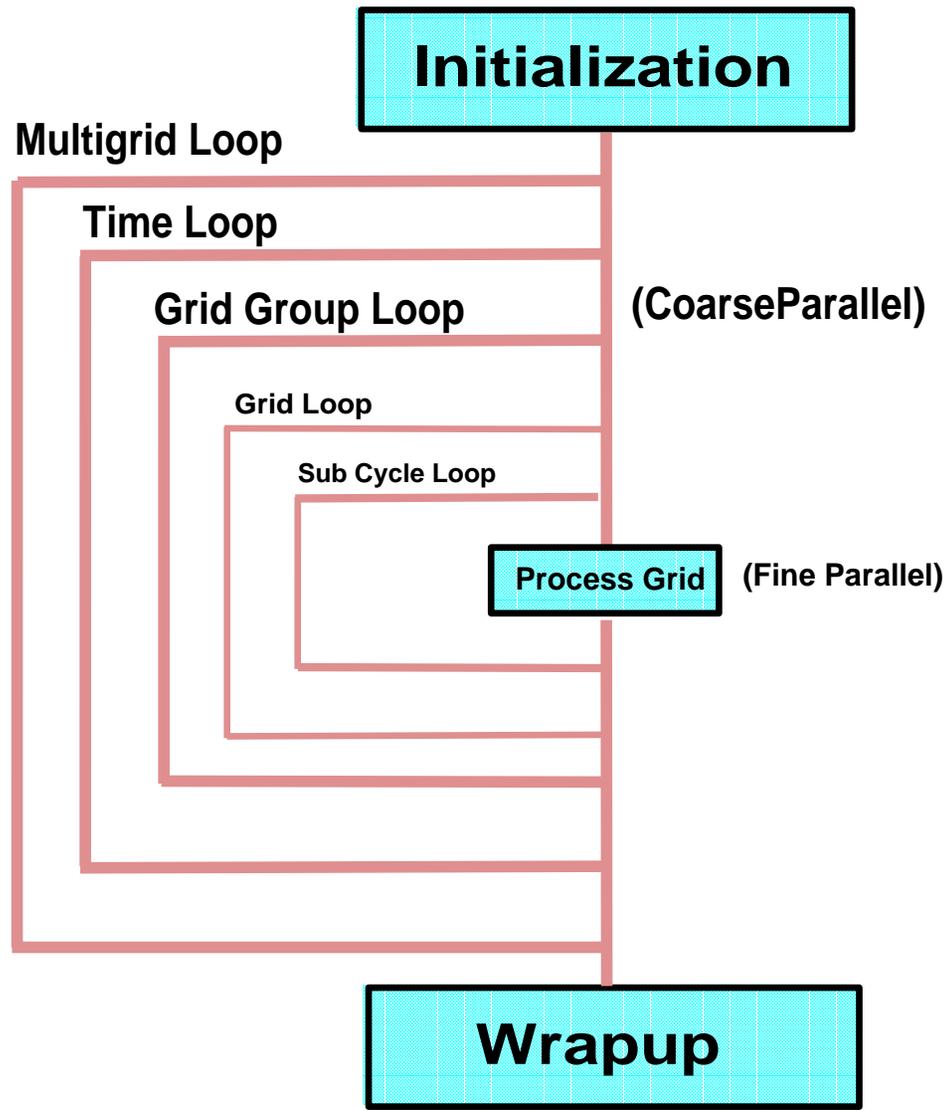
## Drawbacks:

- Parallelism per grid limited

## Results:

- Scales to 11x at best on 16 CPUs

# OVERFLOW – Logic Flow (MLP)



## Characteristics:

- Grid initialization serial
- Grid processing parallel
- Single Grid parallel
- Wrapup serial

## Drawbacks:

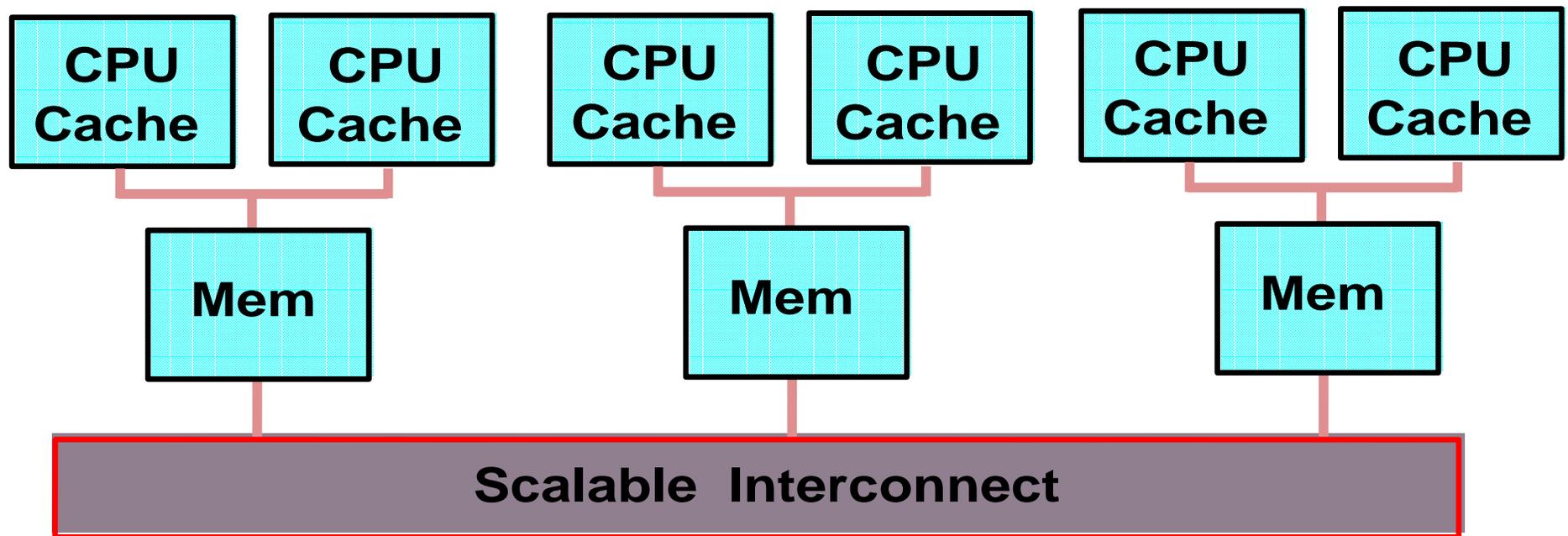
- Parallelism per grid still limited

## Results:

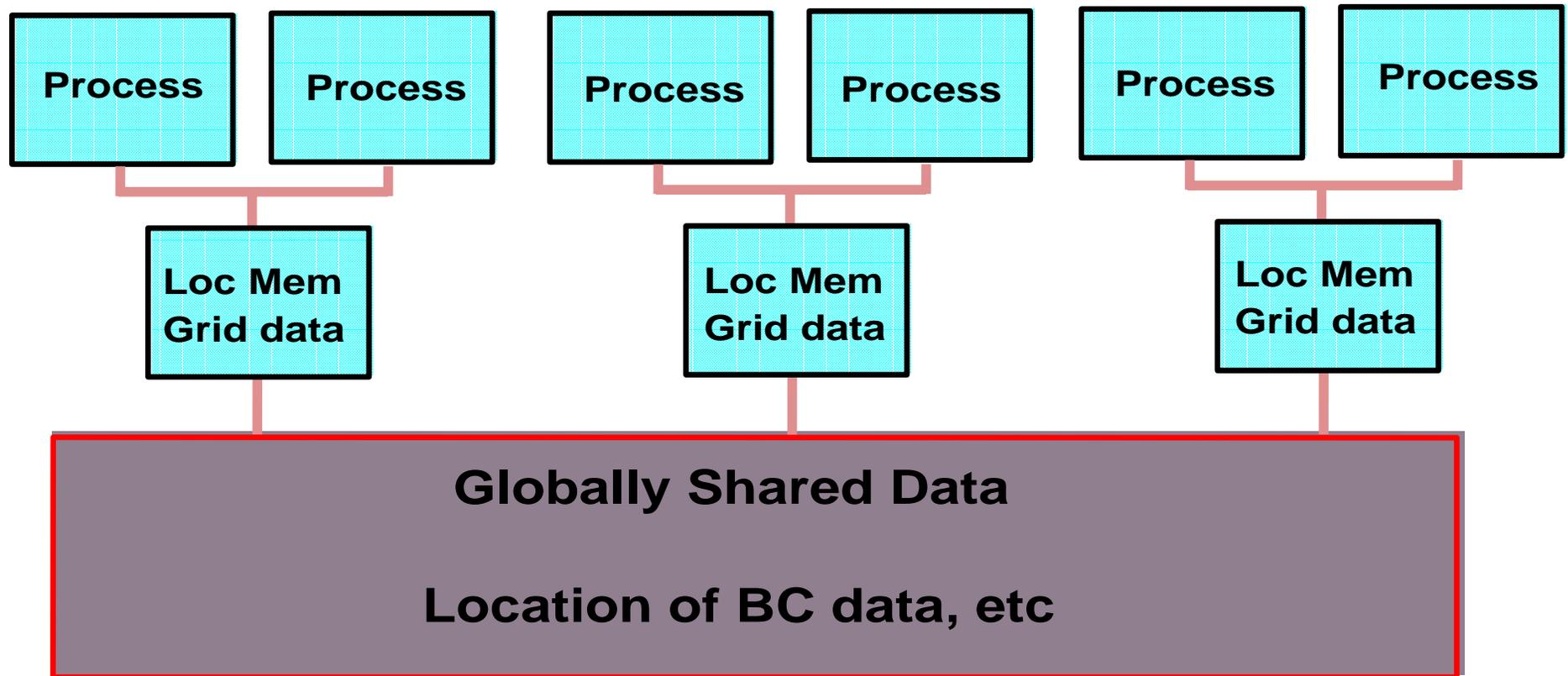
- Scales > 64x on 64 CPUs
- Directly applicable to C90

# *MLP is very NUMA Friendly*

## *Classic NUMA System Architecture*



# Mapping of MLP based code onto Numa System



# ***MLP as Implemented at NAS***

## **Two modes of operation:**

- Case of more grids than CPUs**
  - Groups of grids are mapped onto groups of CPUs**
  - Example: The 153 zone "Airplane" problem on 64 CPUs**
  
- Case of more CPUs than grids**
  - Groups of CPUs mapped onto each grid**
  - Example: The 6 zone wing/body problem on 64 CPUs**
  - Example: The 10 zone 747 problem on 64 CPUs**

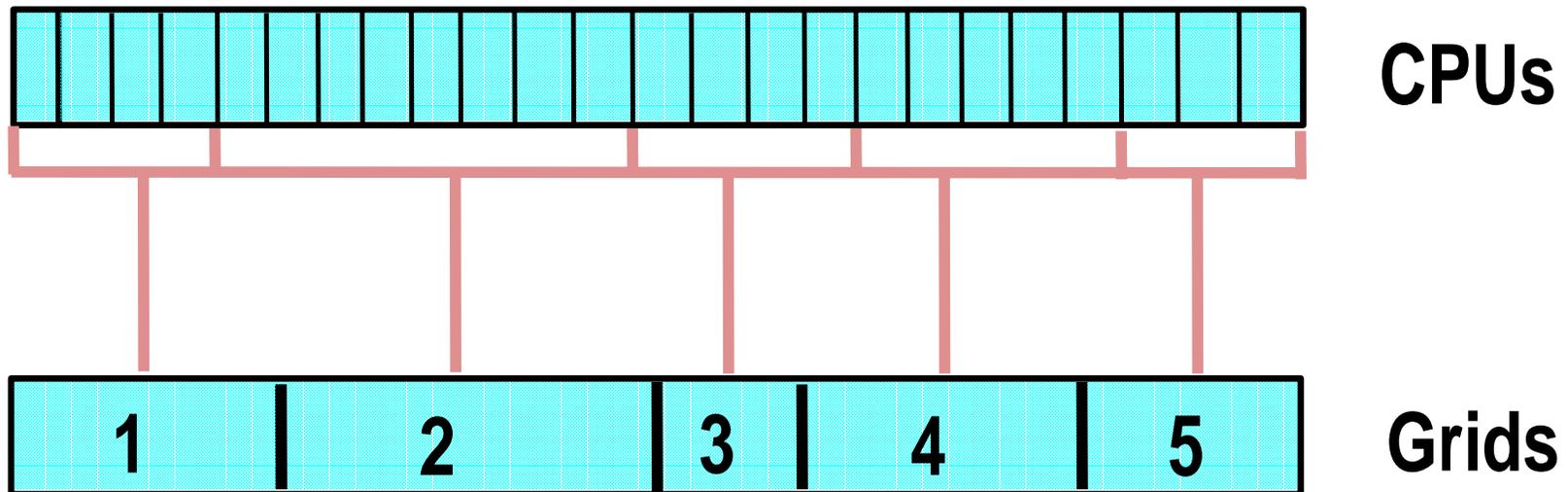
# Mapping the Grids to CPUs

The mapping below shows how a few grids will map to large CPU counts.

The reverse is also true. One CPU could manage many grids.

The most desirable case is that both your grid and cpu count is high

This allows the greatest flexibility in load leveling your workload



# ***Multi Level Paralellism – Summary***

## **Features:**

- Coarse grained parallelism without the aggravation of MPI**
- Decomposition similar to MPI code**
- No messages – share data through shared memory of the SMP**

## **Advantages:**

- No "message" synchronization**
- Major reduction in code complexity**
- Vastly simplified debugging procedure**
- Major increase in scalability as latencies are in nanoseconds**
- Maintains code portability to classic vector systems**
- Maintains code performance on classic vector systems**

## **Disadvantages:**

- Restricted to shared memory systems – but that's the future**

# Typical Profile for OVERFLOW

-----  
Profile listing generated Tue Jun 3 09:45:41 1997  
with: prof BM61.x.pcsamp.11296  
-----

Sorted in descending order by the number of samples in each procedure.  
-----

samples	time(%)	cum time(%)	procedure (dso:file)
3446	34s( 5.3)	34s( 5.3)	SPE2JK (BM61.x:SPE2JK.f)
3433	34s( 5.3)	69s( 10.7)	G2SL (BM61.x:G2SL.f)
3408	34s( 5.3)	1.0e+02s( 16.0)	S2GL (BM61.x:S2GL.f)
2545	25s( 3.9)	1.3e+02s( 19.9)	VFLL (BM61.x:VFLL.f)
2393	24s( 3.7)	1.5e+02s( 23.6)	SCALE (BM61.x:TSCALE.f)
2184	22s( 3.4)	1.7e+02s( 27.0)	TKL (BM61.x:TKL.f)
2167	22s( 3.4)	2.0e+02s( 30.4)	SPE2JL (BM61.x:SPE2JL.f)
2047	20s( 3.2)	2.2e+02s( 33.5)	FLCJ (BM61.x:FLCJ.f)
1855	19s( 2.9)	2.3e+02s( 36.4)	v1pen (BM61.x:v1pen.f)
1747	17s( 2.7)	2.5e+02s( 39.1)	TKINV (BM61.x:TKINV.f)
1639	16s( 2.5)	2.7e+02s( 41.7)	NPINV (BM61.x:NPINV.f)
1587	16s( 2.5)	2.8e+02s( 44.1)	EIGCJ (BM61.x:EIGCJ.f)
1531	15s( 2.4)	3.0e+02s( 46.5)	COPY (BM61.x:COPY.f)
1521	15s( 2.4)	3.2e+02s( 48.9)	v1pen3 (BM61.x:v1pen3.f)
1228	12s( 1.9)	3.3e+02s( 50.8)	BCCHAR (BM61.x:BCCHAR.f)
1183	12s( 1.8)	3.4e+02s( 52.6)	QADDS (BM61.x:QADDS.f)
1078	11s( 1.7)	3.5e+02s( 54.3)	CARK (BM61.x:CARK.f)
1076	11s( 1.7)	3.6e+02s( 55.9)	STFLL (BM61.x:STFLL.f)
1014	10s( 1.6)	3.7e+02s( 57.5)	FLCL (BM61.x:FLCL.f)
957	9.6s( 1.5)	3.8e+02s( 59.0)	STFLK (BM61.x:STFLK.f)
953	9.5s( 1.5)	3.9e+02s( 60.5)	EIGCL (BM61.x:EIGCL.f)
939	9.4s( 1.5)	4.0e+02s( 61.9)	ZEROM (BM61.x:ZEROM.f)
866	8.7s( 1.3)	4.3e+02s( 66.0)	VEIL (BM61.x:VEIL.f)
771	7.7s( 1.2)	4.3e+02s( 67.2)	VSUTH (BM61.x:CVSUTH.f)
722	7.2s( 1.1)	4.4e+02s( 68.3)	LDTJ (BM61.x:LDTJ.f)
700	7s( 1.1)	4.5e+02s( 69.4)	DSEK (BM61.x:DSEK.f)
682	6.8s( 1.1)	4.5e+02s( 70.5)	DFCL (BM61.x:DFCL.f)
681	6.8s( 1.1)	4.6e+02s( 71.5)	v2pen3 (BM61.x:v2pen3.f)

# Typical HPM Counts for OVERFLOW

Based on 195 MHz IP27

Event Counter Name	Counter Value	Typical Time (sec)	Minimum Time (sec)	Maximum Time (sec)
0 Cycles.....	37987705360	194.808745	194.808745	194.808745
16 Cycles.....	37987705360	194.808745	194.808745	194.808745
25 Primary data cache miss.....	2350757904	108.617070	33.995576	108.617070
21 Graduated floating point instructions.....	8475964592	43.466485	21.733243	2260.257225
18 Graduated loads.....	7281846320	37.342802	37.342802	37.342802
2 Issued loads.....	6711397840	34.417425	34.417425	34.417425
26 Secondary data cache misses.....	78964320	30.573365	19.988097	34.015399
22 Quadwords written back from primary data cache.....	1482076256	29.261506	23.865228	33.821740
3 Issued stores.....	2902365184	14.883924	14.883924	14.883924
19 Graduated stores.....	2841032352	14.569397	14.569397	14.569397
23 TLB misses.....	26030144	9.089192	9.089192	9.089192
7 Quadwords written back from scache.....	194797824	6.393364	4.225614	6.393364
6 Decoded branches.....	924424032	4.740636	4.740636	4.740636
9 Primary instruction cache misses.....	14355200	1.326568	0.414460	1.326568
10 Secondary instruction cache misses.....	1013888	0.392557	0.256644	0.436752
24 Mispredicted branches.....	19908528	0.144975	0.065341	0.532936
31 Store/prefetch exclusive to shared block in scache..	4630208	0.023745	0.023745	0.023745
30 Store/prefetch exclusive to clean block in scache..	3090448	0.015848	0.015848	0.015848
4 Issued store conditionals.....	224	0.000001	0.000001	0.000001
20 Graduated store conditionals.....	48	0.000000	0.000000	0.000000
1 Issued instructions.....	40510499216	0.000000	0.000000	207.746150
5 Failed store conditionals.....	0	0.000000	0.000000	0.000000
8 Correctable scache data array ECC errors.....	0	0.000000	0.000000	0.000000
11 Instruction mispredict from scache way pred table..	3295232	0.000000	0.000000	0.016899
12 External interventions.....	313328	0.000000	0.000000	0.000000
13 External invalidations.....	6522864	0.000000	0.000000	0.000000
14 Virtual coherency conditions.....	0	0.000000	0.000000	0.000000
15 Graduated instructions.....	29757330816	0.000000	0.000000	152.601696
17 Graduated instructions.....	29682913856	0.000000	0.000000	152.220071
27 Data misprediction from scache way predict table...	63005824	0.000000	0.000000	0.323107
28 External intervention hits in scache.....	102304	0.000000	0.000000	0.000000
29 External invalidation hits in scache.....	133312	0.000000	0.000000	0.000000

# Typical HPM Statistics for OVERFLOW

## Statistics

```
=====
```

Graduated instructions/cycle.....	0.783341
Graduated floating point instructions/cycle.....	0.223124
Graduated loads & stores/cycle.....	0.266478
Graduated loads & stores/floating point instruction.....	0.021536
Graduated loads/Issued loads.....	1.084997
Graduated stores/Issued stores.....	0.978868
Data mispredict/Data scache hits.....	0.027734
Instruction mispredict/Instruction scache hits.....	0.246995
L1 Cache Line Reuse.....	3.306219
L2 Cache Line Reuse.....	28.769875
L1 Data Cache Hit Rate.....	0.767778
L2 Data Cache Hit Rate.....	0.966409
Time accessing memory/Total time.....	1.027633
L1--L2 bandwidth used (MB/s, average per process).....	507.869771
Memory bandwidth used (MB/s, average per process).....	67.882980
MFLOPS (average per process).....	43.509159
MFLOPS (average per process Assuming MADDs).....	87.018318

To get a better idea of cache line reuse

```
=====
```

L1: 3.306219/8 = 0.41      Each value used 0.41 times after fetch

L2: 28.769875/16 = 1.79      Each value used 1.79 times after fetch

# ***OVERFLOW Results***

***– 10 Zone 747***

***– 6 Zone Wing/Body***

***– 153 Zone Airplane***

# ***Results: The 10 Zone 747 Problem***

**This case was the third case attempted with the OVERFLOW MLP-I code.**

**It is interesting because it has been run using the OVERFLOW MPI code on the Origin, so there are direct comparisons that can be made in timings.**

**The case is a very simple case from a load balance standpoint, as zone sizes vary only about 60%, as opposed to many problems where the zone sizes vary an order of magnitude.**

**This is an MPI friendly test, and should show reasonable scaling on even purely distributed memory architectures like the SP2.**

# Results: 10 Zone 747 Problem - 10 CPUs

## Problem Description:

-----  
Zones 10  
Total Points 2384196  
Largest Zone 279129  
Smallest Zone 172692  
Time Steps 10

## Summary of Load Balance Analysis (volumetric):

-----  
Grid JD KD LD Points % Volume CPUs  
-----  
1 81 41 52 172692 7.24 1  
2 103 39 59 237003 9.94 1  
3 249 19 59 279129 11.71 1  
4 249 19 59 279129 11.71 1  
5 249 19 59 279129 11.71 1  
6 249 18 59 264438 11.09 1  
7 249 18 59 264438 11.09 1  
8 87 41 57 203319 8.53 1  
9 87 41 57 203319 8.53 1  
10 80 56 45 201600 8.46 1  
-----  
Totals: 2384196 100.00 10



# Results: 10 Zone 747 Problem – 64 CPUs

## Problem Description:

-----  
Zones 10  
Total Points 2384196  
Largest Zone 279129  
Smallest Zone 172692  
Time Steps 10

## Summary of Load Balance Analysis

-----  
Group Grid JD KD LD Points % Volume CPUs  
-----  
1 5 249 19 59 279129 11.71 8  
2 4 249 19 59 279129 11.71 8  
3 3 249 19 59 279129 11.71 8  
4 7 249 18 59 264438 11.09 8  
5 6 249 18 59 264438 11.09 7  
6 2 103 39 59 237003 9.94 6  
7 9 87 41 57 203319 8.53 5  
8 8 87 41 57 203319 8.53 5  
9 10 80 56 45 201600 8.46 5  
10 1 81 41 52 172692 7.24 4  
-----

BALANCE: Maximum number of groups set to : 10  
BALANCE: Maximum cpus per group set to : 8  
BALANCE: Maximum cpus in machine set to : 64  
BALANCE: User requests auto distribution of CPUs to groups



# Summary: The 10 Zone 747 Problem

## Summary of OVERFLOW Run results on the 10 Zone Boeing 747 Problem

<u>Code Version</u>	<u>1 CPU</u>	<u>10 CPU</u>	<u>64 CPU</u>
CRAY C90	171		
OVERFLOW SM		380	508
OVERFLOW MPI		165	
OVERFLOW MLP-I		148	33

### Comments:

- 1) An older version of MLP-I runs the 10 cpu case in 127 seconds.
- 2) The 64 CPU MLP-I run suffers from the old core algorithms having no NUMA awareness - changing soon
- 3) Single CPU Cray performance is 447 MFLOPS on this problem

### Notes:

- 1) OVERFLOW SM Standard OVERFLOW 1.7u shared memory C90 code
- 2) OVERFLOW MPI Jespersen's MPI version of OVERFLOW
- 3) OVERFLOW MLP-I OVERFLOW SM with MLP parallelism added (no single zone optimization)

# ***Results: The 6 Zone Wing/body Problem***

**This case was the second case attempted with the OVERFLOW MLP-I code. The grid was suggested and provided by Dennis Jespersen who had been using it to test grid sub-division in the OVERFLOW MPI code. Comparisons with those results are provided herein.**

**This problem is interesting because of the large variation in grid sizes. It is a good test of the robustness of the load leveling capabilities of the OVERFLOW MLP code.**

# Results: 6 Zone Wing/Body Problem

## Problem Description:

-----

Zones 6  
Total Points 1061574  
Largest Zone 418779  
Smallest Zone 8959  
Time Steps 30

## Summary of Load Balance Analysis (Volumetric):

Grid	JD	KD	LD	Points	% Volume	CPUs
2	237	57	31	418779	39.45	1
4	141	57	37	297369	28.01	1
3	237	25	31	183675	17.30	1
1	111	27	31	92907	8.75	1
6	59	29	35	59885	5.64	1
5	17	17	31	8959	0.84	1
Totals:				1061574	100.00	6



# Results: 6 Zone Wing/Body Problem – 6 CPUs/1 GRP

## Net OVERFL Workload by Group

```

-----
100 % * x
      * x
      * x
      * x
      * x
      * x
      * x
      * x
      * x
      * x
      * x
75 % * x
     * x
     * x
     * x
     * x
     * x
     * x
W    * x
A    * x
L    * x
T    * x
I 50 % * x
M    * x
     * x
     * x
     * x
     * x
     * x
     * x
     * x
     * x
     * x
25 % * x
     * x
     * x
     * x
     * x
     * x
     * x
     * x
     * x
     * x
****
      1
      Group Number
  
```

## Raw Data Summary

Group	OVERFL	BARRIER	TOTAL
1	350.54	0.00	350.54

## Summary of Load Balance Analysis: Group Number: 1

Grid	JD	KD	LD	Points	% Volume	CPUs
2	237	57	31	418779	39.45	6
4	141	57	37	297369	28.01	6
3	237	25	31	183675	17.30	6
1	111	27	31	92907	8.75	6
6	59	29	35	59885	5.64	6
5	17	17	31	8959	0.84	6
<b>Totals:</b>				1061574	100.00	6

## Notes

- 1) Total of 1 Group in this run with 6 CPUs per group
- 2) Load leveling does not apply here as there is only 1 group



# Results: 6 Zone Wing/Body Problem – 16 CPUs/2 GRPs

## Net OVERFL Workload by Group

```

-----
100 % * x x
      * x x
      * x x
      * x x
      * x x
      * x x
      * x x
      * x x
      * x x
      * x x
      * x x
75 % * x x
     * x x
     * x x
     * x x
     * x x
     * x x
     * x x
     * x x
     * x x
     * x x
     * x x
50 % * x x
     * x x
     * x x
     * x x
     * x x
     * x x
     * x x
     * x x
     * x x
     * x x
     * x x
25 % * x x
     * x x
     * x x
     * x x
     * x x
     * x x
     * x x
     * x x
     * x x
     * x x
*****
      1 2
      Group Numbe
  
```

## Raw Data Summary

Group	OVERFL	BARRIER	TOTAL
1	155.45	2.64	158.09
2	158.09	0.00	158.09

### Summary of Load Balance Analysis: Group Number: 1

Grid	JD	KD	LD	Points	% Volume	CPUs
2	237	57	31	418779	39.45	8
1	111	27	31	92907	8.75	8
5	17	17	31	8959	0.84	8
Totals:				1061574	49.04	8

### Summary of Load Balance Analysis: Group Number: 2

Grid	JD	KD	LD	Points	% Volume	CPUs
4	141	57	37	297369	28.01	8
3	237	25	31	183675	17.30	8
6	59	29	35	59885	5.64	8
Totals:				1061574	50.96	8

## Notes

- 1) Total of 2 Groups in this run with 8 CPUs each
- 2) Terrific load leveling





# Summary: 6 Zone Wing/body Problem

## Summary of OVERFLOW Run results on the 6 Zone Wing/Body Problem

### TIMES:

-----

<u>Code Version</u>	<u>1 CPU</u>	<u>6 CPU</u>	<u>8 CPU</u>	<u>17 CPU</u>	<u>32 CPU</u>	<u>64 CPU</u>
CRAY C90	223					
OVERFLOW SM	1240					
OVERFLOW MPI			266	210		
OVERFLOW MLP-I (6gr)		515		149 (16cpu)	83	62
OVERFLOW MLP-I (2gr)				158 (16cpu)		
OVERFLOW MLP-I (1gr)		350				

### Comments:

- 1) Scaling past 32 CPUs is hampered by old code limited to 8 CPU scaling and lack of NUMA awareness
- 2) This will be fixed. Preliminary results are encouraging
- 3) Times are for the OVERFL routine and in wall clock seconds
- 4) Single CPU Cray performance is 467 MFLOPS on this problem

### Notes:

- 1) OVERFLOW SM Standard OVERFLOW 1.7u shared memory C90 code
- 2) OVERFLOW MPI Jespersen's MPI version of OVERFLOW
- 3) OVERFLOW MLP-I OVERFLOW SM with MLP parallelism added (no single zone optimization)

# ***OVERFLOW Results – The "Airplane" Problem***

**The "Airplane" problem is a real world problem addressed by the ATDE**

**The customer needs to quickly obtain results from codes like OVERFLOW during the course of a wind tunnel experiment.**

**These results need to be timely so that rapid reconfiguration of the experiment is possible.**

**NAS has committed to support this effort by providing dedicated C90 time in 48 hour parcels.**

**Needless to say dedication of such compute resources is extremely costly and impacts the availability of resources to the rest of the community.**

**Another solution needed to be found...**

# ***Results: The "Airplane" Problem – 63 CPUs***

**This case was the third case attempted with the OVERFLOW MLP code. The grid was provided by Karlin Roth and Yehia Rizk. It is a 33 million point problem distributed across 153 grids of widely varying size.**

**This problem is interesting for several reasons. They are:**

- It is one of the largest problems ever solved at NAS**
- It requires hundreds of fully dedicated C90 hours per run**
- It fully stress tests the MLP code**
- It fully stress tests the Origin system components**

# Results: 153 Zone Airplane Problem – 63 CPUs

Problem Description:

```

-----
Zones                153
Total Points         33126590
Largest Zone         1414140
Smallest Zone        23625
Time Steps           10
  
```

Summary of Load Balance Analysis: Group Number: 1

```

-----
Grid   JD   KD   LD   Points  % Volume  CPUs
-----
   6   111  140  91   1414140  4.27     3
   92   35   41   51    73185   0.22     3
   36   43   31   41    54653   0.16     3
  107   29   31   31    27869   0.08     3
-----
Totals:                33126590  4.74     3
  
```

Summary of Load Balance Analysis: Group Number: 21

```

-----
Grid   JD   KD   LD   Points  % Volume  CPUs
-----
   98   205   37   51   386835  1.17     3
  152   95   65   51   314925  0.95     3
    2  115   52   42   251160  0.76     3
   27  303   16   45   218160  0.66     3
  143   73   44   45   144540  0.44     3
   97   42   49   51   104958  0.32     3
  130   81   19   45    69255  0.21     3
  141   69   21   31    44919  0.14     3
  140   35   29   33    33495  0.10     3
-----
Totals:                33126590  4.73     3
  
```











# Summary: 153 Zone Airplane Problem

## Comparisons of MFLOP Ratings

---

<u>Machine</u>	<u>Elapsed Time</u>	<u>16 CPU MFLOPS</u>
Cray C90	463 sec	4.62 GFLOPs (28% of peak)
Origin	642 sec	3.33 GFLOPs (13% of peak)
Origin (assume levler)	590 sec	3.62 GFLOPS

## Comparison of I/O Wait

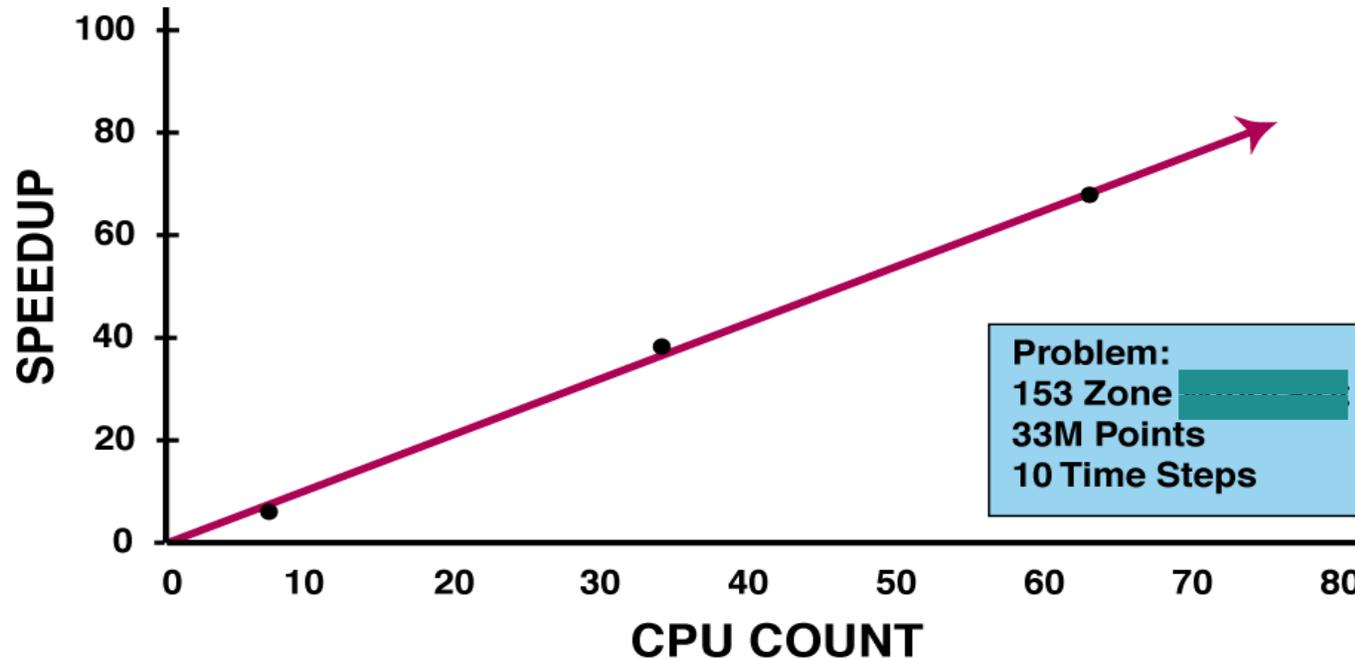
---

<u>Machine</u>	<u>Elapsed Time</u>
Cray C90	168 sec
Origin	160 sec

**NOTE: Origin GFLOPS based on Cray FLOP count and Origin time.**

# *Do we have a scaling problem?*

## OVERFLOW MLP-I SCALING (SPEEDUP VS. CPU COUNT)



# ***Origin 2000 Project Highlights***

- Origin 2000 absolute performance is comparable to full C90**
  - ARC3D performance is 118% of full C90**
  - OVERFLOW performance is 72% of full C90**
- Striking Cost/Performance Advantage:**
  - 2.5M/1.18 vs 25M/1.00 = 11.8x for ARC3D**
  - 2.5M/0.71 vs 25M/1.00 = 7.1x for OVERFLOW**
- Dramatic performance gains for very small changes in code**
  - ~1000 lines in ARC3D**
  - ~ 200 lines in OVERFLOW**
- Potential for much greater performance and scaling**
  - Airplane problem could scale to 512 Processors**
  - If true, that's ~ 5.76 x the performance of full C90**

# ***Origin 2000 Project Accomplishments***

**There are a number of major milestones that have been accomplished:**

- Dramatically improved scaling for ARC3D. This was obtained using classic coarse and fine grained parallelism on a Origin 2000 system with little work.**
- Dramatically improved scaling for OVERFLOW was obtained using state of the art Multi-Level Shared Memory Parallelism – A first in the nation for production CFD**
- A new and highly successful dynamic load balancing facility has been added to OVERFLOW to greatly enhance its robustness for future work.**
- The work has pointed the way to a standardized method for developing CFD code for the future that both, preserves the high performance of the code on classic vector machines, yet takes advantage of the new RISC/NUMA system features as well.**
- Finally, the effort proves that highly complex code can be successful on NUMA non-vector systems with modest efforts.**

# ***Future Directions***

## **– OVERFLOW Work**

- Single CPU optimizations**
- Better memory locality optimizations**

## **– Need for Larger System – 256 CPUs**

- Potential to achieve 30 GFLOPS (IT/ACNS milestone)**
- "Airplane" problem turnaround in hours not days**
- Would permit scheduling 64 and 128 CPU jobs.**
  - Larger system essential to support this routinely**
  - This kind of access critical to code development**

## **– SGI best fit for now**

- Larger system architecturally identical to current system**
- 64 Bit Operating System – critical for large problems**

# ***The State of Parallel Processing at ARC/NAS***

**ARC is currently a national leader in effective use of NUMA**

- ARC is the only site with a NUMA aware batch scheduler that addresses throughput issues found on these systems**
- ARC is the only site with a successful implementation of MLP and has the opportunity to set the standard in this area of research.**
- ARC has the opportunity to become the leader within NASA for evolving the engineering codes of today into the NUMA world of tomorrow.**

# ***Some comments on Overflow***

- The code is not NUMA aware in any sense and poor memory locality for the big problem dramatically slows it down. This is akin to Cray memory bank conflicts.**
- The code makes little use of L1 cached data with a serious reduction in performance.**
- Within the next two months the code will execute the large problem at 2x over current speeds. Within 6 months we can expect an additional 2x.**
- SGI will be announcing a mid-life kicker at some point in the not too distant future. The end result will be dramatic reduction in the code run times with no changes at all.**